SETTING UP FOR THE PROJECT

Step 1: Create the database:

1) Open an EXISTING database connection in DbVisualizer.
2) Run the command **`CREATE DATABASE birthdaybook;`**
3) Close the existing connection.
4) Create a DbVisualizer connection for the birthdaybook database and connect to it.
5) Once your NEW connection is open, open the file
   `sample-birthday-book-project\database\birthdaybook.sql`
6) Execute the script to create the database and a few records;

*****NOTE: The database name MUST be birthdaybook in order for tests to work correctly*****

Step 2: Import the project into Eclipse.

Step 3: Before you start coding, confirm your setup is correct by:

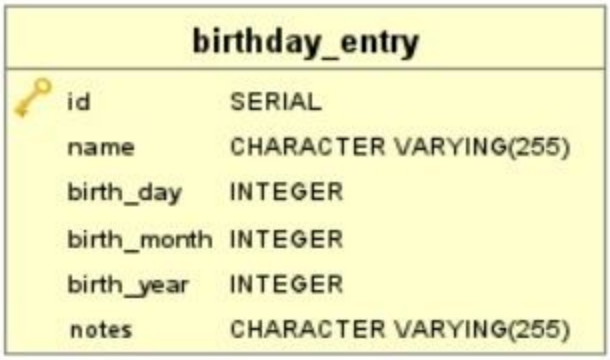
1) Open the file **`JDBCBirthdayEntryDAOTest`** class in the **`com.techelevator.birthdaybook.dao package`** of the **`src/test/java folder`** .
2) Run the test **`create_withValidData_shouldCreateRecord`**
3) All the other tests in this class will fail but if your setup is correct this one will pass.

Step 4: Your setup is COMPLETE!!!!!

PART 1: The DAO Code

This project is a “birthday book” - basically an app that lets you store information about people and their birthdays. The data model is a bit contrived but it’s done this way to allow for practice of various topics you have learned.

The data model looks like this:



Step 1: Start with completing the database code:

The **`JDBCBirthdayEntryDao`** class in the **`com.techelevator.birthdaybook.dao`** package has been provided for you.

Currently, the **`create`** method is completed… it is up to you to fill in the rest of the methods:

1) Start with the **`getEntries()`** method:
    a) Write the code to get all the **`BirthdayEntry`** objects in the database.
    b) The method **`mapRowToBirthdayEntry`** to map a **`SqlRowSet`** row to a **`BirthdayEntry`** has been provided but you will need to complete it in order to be able to use it.
    c) If you complete this correctly, the **`getEntries_withValidData_shouldReturnMultipleRecordsin`** the **`JDBCBirthdayEntryDAOTest`** class should pass.

2) Next, complete the **`getEntry(Long id)`** method:
    a) Write the code to get a specific **`BirthdayEntry`** record by id.
        i) **`getEntry_withValidId_shouldReturnRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
    b) In the case that the record is not found, throw an **`EntryNotFoundException`** (this class has been provided but note that you will need to make some changes to the method signature in order to accomplish this).
        i) **`getEntry_withValidId_shouldReturnRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.

3) Next, complete the **`deleteEntry(Long id)`** method.
    a) Write the code to delete a specific BirthdayEntry record by id.
        i) **`deleteEntry_withValidId_shouldDeleteRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
    b) In the case that the record is not found, throw an **`EntryNotFoundException`**.
        i) In order to be able to know whether the record was deleted or not, you will need to check the number of records affected. The **`jdbcTemplate.update`** method returns a count of affected rows as an integer (we haven't used this value so far but it does actually return a count and you can assign the return value to an int variable) and if that count is 0, the record did not exist.
        ii) **`deleteEntry_withInValidId_shouldThrowExceptionin`** in the **`JDBCBirthdayEntryDAOTest`** class should pass.
        iii) **`deleteEntry_withValidId_shouldNotThrowException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

4) Next, complete the **`updateEntry(BirthdayEntry entry, Long id)`** method.
    a) Write the code to update a specific **`BirthdayEntry`** record by id.
        i) The basic skeleton has been provided. Your query should go where the comment **`// update here`** is.
        ii) **`updateEntry_withValidData_shouldUpdateRecord`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.
    b) If the record does not exist, a **`DataAccessException`** will be thrown by the system. The provided code catches this exception but when it does, you should throw a **`EntryNotFoundException`** (this essentially changes the exception to beone related to your code rather than the systemone).
        i) **`updateEntry_withInvalidData__shouldThrowEntryNotFoundException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

c) The skeleton code checks to make sure that the id in the **`BirthdayEntry`** and the id provided match. If the scenario where the ids don't match occurs, you should also throw a **`EntryNotFoundException`**.
    i) **`updateEntry_withInvalidRecord_shouldThrowEntryNotFoundException`** in the **`JDBCBirthdayEntryDAOTest`** class should also pass.

At this point, all the tests in **`JDBCBirthdayEntryDAOTest`** should pass.

Congrats… you have finished Part 1!

PART 2: DTO Validation

Currently, the **`BirthdayEntry`** class is set up to handle validation, but it has no rules in place to be used for validation At the moment, only the **`should_have_no_violations`** test in the **`BirthdayEntryValidationTests`** class should pass.

Add the following rules to the **`BirthdayEntry`** DTO:

1) The **`name`** field should not be blank. Add a rule to check for this and use the error message **`"The field name should not be blank."`** if this rule is violated.
    a) If you complete this change correctly, the **`nameShouldNotBeBlanktest`** test in the **`BirthdayEntryValidationTests`** should pass.

2) The **`birthDay`** field should be in the range of 1 to 31 .Add a rule to check for this and use the error message **`"The field birth day must be at least 1"`** if the value is less than one or **`"The field birth day must be less than or equal to 31"`** if the value is more than 31.
    a) If you complete this change correctly, the **`birthDayShouldBe1OrMore`** and **`birthDayShouldBe31OrLess`** tests the **`BirthdayEntryValidationTests`** should pass.

3) The **`birthMonth`** field should be in the range of 1 to 12. Add a rule to check for this and use the error message **`"The field birth month must be at least 1"`** if the value is less than one or **`"The field birth month must be less than or equal to 12"`** if the value is more than 12.
    a) If you complete this change correctly, the **`birthMonthShouldBe1OrMore`** and **`birthMonthShouldBe12OrLess`** tests the **`BirthdayEntryValidationTests`** should pass.

4) The **`birthYear`** field should be positive. Add a rule to check for this and use the error message **`"The field birth year must be positive."`** if this rule is violated.
    a) If you complete this change correctly, the **`nameShouldNotBeBlanktest`** test iin the **`birthYearShouldBePositive`** should pass.

At this point all tests in the **`BirthdayEntryValidationTests`** class (including the original **`should_have_no_violations`** test)should pass.

Congrats… you have finished Part 2!

PART 3: Wiring up the API

In this final part, you will wire up the controller to allow API access to all the methods you have created.

Currently the **`BirthdayBookController`** has the method to get the list of all objects using a **`GET`** HTTP method. You should see the **`getEntries_withData_shouldReturnData`** test in the **`BirthdayBookControllerTests`** class pass. You may see a few others pass but they may change as you add code so don't pay attention to their status until they are mentioned with the corresponding additions to **`BirthdayBookController`**.

Before you begin working with the controller;

Our controller will handle the **`EntryNotFoundException`** which may be thrown by our DAO and return the correct HTTP status code (**404 Not Found**) when this exception is thrown. In order for this to work correctly in the controller, you will need to modify the **`EntryNotFoundException`** class so that it causes this status to be returned when it is thrown.

NOTE: I had an issue where my Eclipse couldn't see the package containing HttpStatus in IntelliSense in the `EntryNotFoundException` class . If this happens to you, you should be able to manually add this to your imports section to resolve the issue:

```
import org.springframework.http.HttpStatus;
```

Add the following capabilities to the **`BirthdayBookController:`**

1) Get a specific **`BirthdayEntry`** by id.
    a) Write the API code to get a specific **`BirthdayEntry`** by id.
    b) You will need to allow the controller to handle the **`EntryNotFoundException`** which may be thrown by the DAO
    c) If you complete this correctly, the **`getEntry_withValidId_shouldReturnData`** and the **`getEntry_withInvalidId_shouldReturnNotFoundStatus`** tests in the **`BirthdayBookControllerTests`** class should pass.
    d) You should also be able to test the API from Postman while your app is running.

2) Add a **`BirthdayEntry`** record.
    a) Write the API code to add a **`BirthdayEntry`**.
    b) The code should return a **201 CREATED** HTTP status code if the record is created successfully.
        i) If you complete this correctly, the **`createEntry_withValidData_shouldCreateRecord`** in the **`BirthdayBookControllerTests`** class should pass.
    c) Modify your method to validate the **`BirthdayEntry`** being passed in to the method meets all the requirements.
        i) If you complete this correctly, the **`createEntry_withInvalidData_shouldReturnBadRequest`** in the **`BirthdayBookControllerTests`** class should pass
    d) You should also be able to test the API from Postman while your app is running. (You may need some help figuring out how to do this if you haven't used Postman to post info before).

3) Update a **`BirthdayEntry`** record.
    a) Write the API code to update a **`BirthdayEntry`**. The endpoint should accept both a **`BirthdayEntry`** object and the **`id`** of the object to be updated (this is needed by the DAO code you wrote).
    b) You will need to allow the controller to handle the **`EntryNotFoundException`** which may be thrown by the DAO.
        i) If you complete this correctly, the **`update_withValidData_shouldReturnData`** and the **`update_withInvalidData_shouldReturnNotFound`** tests in the **`BirthdayBookControllerTests`** class should pass. (The name of this test is not specific enough… this tests the case in which there is no existing record to update in the database.)
    c) Modify your method to validate the **`BirthdayEntry`** being passed in to the method meets all the requirements.
        i) If you complete this correctly, the **`update_withInvalidData_shouldReturnBasdRequst`** and the **`BirthdayBookControllerTests`** class should pass
    d) You should also be able to test the API from Postman while your app is running. (You may need some help figuring out how to do this if you haven't used Postman to post info before).
4) Delete a **`BirthdayEntry`** record.
    a) Write the API code to delete a **`BirthdayEntry`**.
    b) The code should return a **204 NO CONTENT** HTTP status code if the record is created successfully.
    c) You will need to allow the controller to handle the **`EntryNotFoundException`** which may be thrown by the DAO.

i) If you complete this correctly, the **`delete_withValidId_shouldReturnNoContent`** and the **`delete_withInvalidId_shouldReturnNotFound`**tests in the **`BirthdayBookControllerTests`** class should pass.

d) You should also be able to test the API from Postman while your app is running..

At this point all the tests for the project should pass. If they do…

You have successfully completed the project - CONGRATS!